**1、本章面试题**

**2、知识点**

**2.1、课程回顾**

**2.2、本章重点**

**3、具体内容**

**拦截器**

**1)编写自定义拦截器  LoginInterceptor**

**2)在入口类的目录或者兄弟目录下创建一个类继承WebMvcConfigurerAdapter类并重写      addInterceptors方法；**

@SpringBootConfiguration   //注解表明这是一个配置类

public class SpringMVCConfigClass extends WebMvcConfigurerAdapter {

    @Override

    public void addInterceptors(InterceptorRegistry registry) {

        //注册自定义拦截器，添加拦截路径和排除拦截路径

        registry.addInterceptor (new LoginInterceptor()).addPathPatterns("/\*\*").excludePathPatterns("/hell");

    }

}

**配置静态文件目录和模板文件目录**

<https://www.thymeleaf.org/index.html>

**1)创建文件夹**

静态文件目录：src\main\resources\static

模板文件目录：src\main\resources\templates

**2）添加依赖包**

<!--模板thymeleaf包-->

    <dependency>

      <groupId>org.springframework.boot</groupId>

      <artifactId>spring-boot-starter-thymeleaf</artifactId>

    </dependency>

**3）添加配置**

 #配置静态资源前后缀

spring.thymeleaf.prefix=classpath:/templates/

spring.thymeleaf.suffix=.html

#spring.thymeleaf.mode=HTML5

#spring.thymeleaf.encoding=UTF-8

#spring.thymeleaf.servlet.content-type=text/html

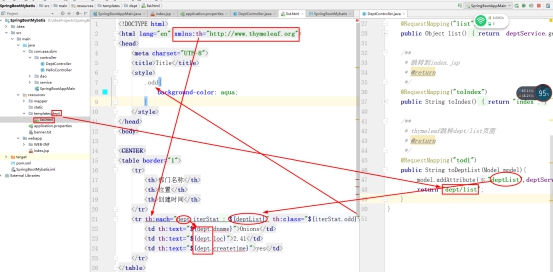
#开发时建议将spring.thymeleaf.cache设置为false，否则会有缓存，导致页面没法及时看到更新后的效果

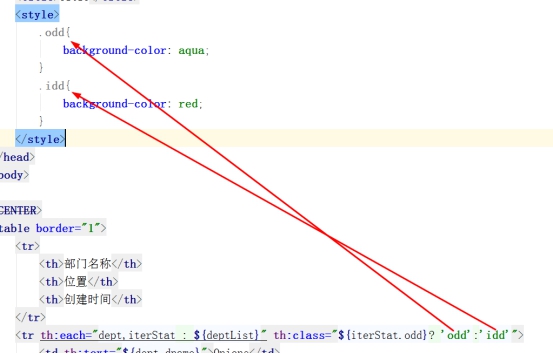
#spring.thymeleaf.cache=false

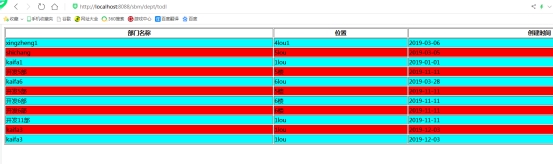
4）编写方法，类似jsp模板，绑定数据传参



   5)编写页面，使用thymeleaf标签



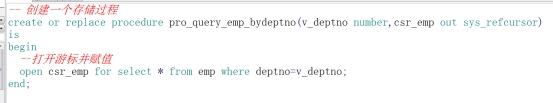




<https://www.cnblogs.com/topwill/p/7434955.html>

**SM整合调用存储过程**

1）编写存储过程



2）编写Mapper配置文件（需要使用实体）

 <!--调用存储过程 根据部门ID或者部门名称  -->

  <select id="queryDnamebyPro" parameterType="Map" statementType="CALLABLE" >

      <!-- statementType="CALLABLE" 调用函数或者存储过程必须配置 -->

      call pro\_query\_deptname\_by\_deptno(#{deptno,jdbcType=DECIMAL,mode=IN},#{dname,jdbcType=VARCHAR,mode=OUT})

  </select>

  <!--调用存储过程 根据部门ID获取部门列表  -->

  <select id="queryDeptsbyPro" parameterType="Map" statementType="CALLABLE" >

      <!-- statementType="CALLABLE" 调用函数或者存储过程必须配置 -->

      call pro\_query\_depts\_by\_deptno(#{deptno,jdbcType=DECIMAL,mode=IN},#{dlist,resultMap=deptList,jdbcType=CURSOR,mode=OUT,javaType=java.sql.ResultSet})

  </select>

  <resultMap type="com.aaa.ssm.entity.Dept" id="deptList">

      <id column="deptno" property="deptno"/>

      <result column="dname" property="dname"/>

      <result column="loc" property="loc"/>

  </resultMap>

3）服务层调用

  deptDao.queryDnamebyPro(map);

return map.get("dname")+"";

deptDao.queryDeptsbyPro(map);

List<Dept> deptList= (List<Dept>)map.get("dlist");

if(deptList!=null&&deptList.size()>0)

return deptList;

return null;

 **3.3 上传**

 1）#配置上路径

             upload.path=D:/images/

#默认支持文件上传

spring.servlet.multipart.enabled=true

spring.servlet.multipart.file-size-threshold=0

#上传文件大小配置

spring.servlet.multipart.max-file-size=10mb

spring.servlet.multipart.max-request-size=10mb

       2）编写页面，注意 method="post" enctype="multipart/form-data"

3）编写controller 修改一起的上传方法，固定硬盘位置  注意@RequestParam MultipartFile mfile  其中mfile和页面file的name对应

       4） 编写显示方法

private final ResourceLoader resourceLoader;

    @Autowired

    public DeptAController(ResourceLoader resourceLoader) {

        this.resourceLoader = resourceLoader;

    }

    @Value("${upload.path}")

    private String path;

   @RequestMapping("show")

    public ResponseEntity show(String fileName){

        try {

            // 由于是读取本机的文件，file是一定要加上的， path是在application配置文件中的路径

            return ResponseEntity.ok(resourceLoader.getResource("file:" + path + fileName));

        } catch (Exception e) {

            return ResponseEntity.notFound().build();

        }

    }

**mybatis分页插件（非官网）**

**1，添加jar包**

        <!-- 分页插件pagehelper -->        <dependency>            <groupId>com.github.pagehelper</groupId>            <artifactId>pagehelper</artifactId>            <version>5.0.0</version>        </dependency>        <dependency>            <groupId>com.github.pagehelper</groupId>            <artifactId>pagehelper-spring-boot-autoconfigure</artifactId>            <version>1.2.3</version>        </dependency>        <dependency>            <groupId>com.github.pagehelper</groupId>            <artifactId>pagehelper-spring-boot-starter</artifactId>            <version>1.2.3</version>        </dependency>      <!-- 分页插件pagehelper -->

**2，添加springboot配置**

[**https://pagehelper.github.io/**](https://pagehelper.github.io/)

[**https://github.com/pagehelper/Mybatis-PageHelper/blob/master/README\_zh.md**](https://github.com/pagehelper/Mybatis-PageHelper/blob/master/README_zh.md)

 #分页插件

#helperDialect属性来指定分页插件使用哪种方言

pagehelper.helper-dialect=oracle

#当该参数设置为 true 时，pageNum<=0 时会查询第一页， pageNum>pages（超过总数时），会查询最后一页。

pagehelper.reasonable=true

#支持通过 Mapper 接口参数来传递分页参数，默认值false，分页插件会从查询方法的参数值中，自动根据上面 params 配置的字段中取值，查找到合适的值时就会自动分页。

pagehelper.support-methods-arguments=true

#增加了该参数来配置参数映射，用于从对象中根据属性名取值

pagehelper.params=count=countSql

**3，具体使用**

   与以前controller区别

  //设置当前第几页和每页显示数量    PageHelper.startPage(Integer.valueOf(map.get("pageNo")+""),Integer.valueOf(map.get("pageSize")+""));

        //用PageInfo对结果进行包装

        PageInfo<Map> pageInfo =new PageInfo<Map>(newsService.getList());

/\*\*

     \* 分页部门查询

     \* @param map

     \* @return

     \*/

    @ResponseBody

    @RequestMapping("page")

    public Object page(@RequestParam Map map){

        int pageNo = Integer.valueOf(map.get("pageNo")+"");

        int pageSize = Integer.valueOf(map.get("pageSize")+"");

        //初始化配置

        PageHelper.offsetPage(pageNo,pageSize);

        PageInfo<Map> pageInfo = new PageInfo<Map>(deptService.getList());

       //如果使用easyui可以这样封装,其他框架，自己根据pageInfo解析

        Map tmap  = new HashMap();

        tmap.put("total",pageInfo.getTotal());

        tmap.put("rows",pageInfo.getList());

        return tmap;

    }

**springboot访问templates配置**

spring.resources.static-locations=classpath:/templates/

**springboot 启动执行sql语句**

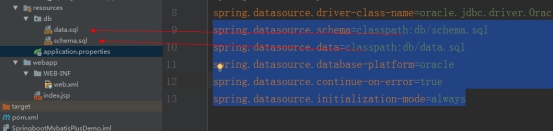
spring.datasource.schema=classpath:db/schema.sql

spring.datasource.data=classpath:db/data.sql

spring.datasource.database-platform=oracle

spring.datasource.continue-on-error=true

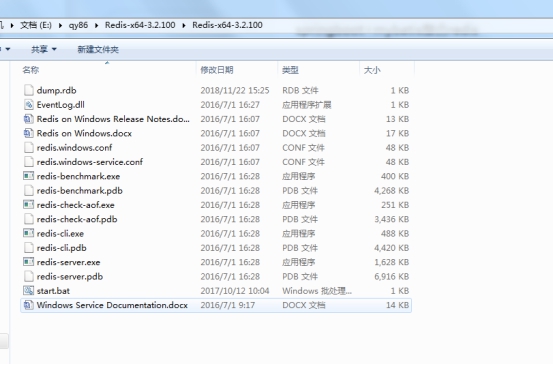
spring.datasource.initialization-mode=always

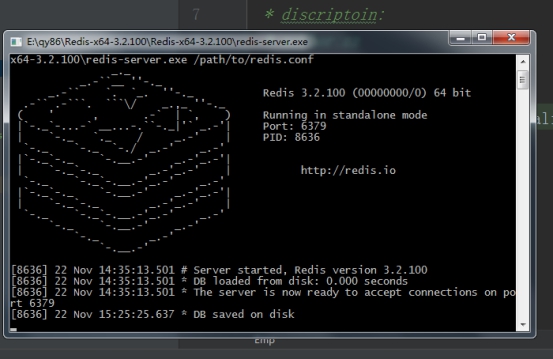


**mybatis整合redis**

**单机版**

安装redis 测试是否能用





ping  测试服务器是否通

 set  aaa 111  设置

 get aaa   取值

mset bbb 222  ccc 333  批量设置

mget aaa ccc     批量获取

del   aaa  删除单个

 flushall  清空所有

  keys  \*  查看所有

shutdown save/nosave 保存/不保存退出

pom文件引入jar包

   <!-- jedis包 -->

    <dependency>

      <groupId>redis.clients</groupId>

      <artifactId>jedis</artifactId>

      <version>2.8.0</version>

    </dependency>

    <!-- mybatis 整合redis 包 -->

    <dependency>

      <groupId>org.mybatis.caches</groupId>

      <artifactId>mybatis-redis</artifactId>

      <version>1.0.0-beta2</version>

    </dependency>

在resources下创建redis.properties

#redis的服务器地址

host=192.168.23.30

#redis的服务端口

port=6379

#链接数据库

default.db=0

#客户端超时时间单位是毫秒

timeout=60000

#最大连接数

maxActive=300

#最大空闲数

maxIdle=100

#最小空闲数

minIdle=1

#最大建立连接等待时间

maxWait=1000

#指明是否在从池中取出连接前进行检验,如果检验失败,则从池中去除连接并尝试取出另一个

testOnBorrow=true

#当调用return Object方法时，进行有效性检查

testOnReturn=true

在mapper文件 或者dao接口上加入缓存配置

<cache type="org.mybatis.caches.redis.RedisCache" blocking="false"

           flushInterval="0" readOnly="true" size="1024" eviction="FIFO"/>

@CacheNamespace(implementation = RedisCache.class)

注意点： 如果使用实体，实体必须序列化：implements Serializable

               退出并保存数据,客户端执行：shutdown SAVE

               解决Springboot使用Redis反序列化遇到的类型转换异常

<https://www.jianshu.com/p/b7635b08a6ce?utm_source=oschina-app>

                删除热部署

**集群版：**

<https://gitee.com/loveliyiyi/mybatis-redis>

#redis的服务器地址

#host=192.168.23.30

#redis的服务端口

#port=6379

#redis地址，多个以逗号分割

address=192.168.23.30:7001,192.168.23.30:7002,192.168.23.30:7003,192.168.23.31:7004,192.168.23.31:7005,192.168.23.31:7006

#部署类型，0，单机；1，分片；2，集群。

deployType=2

#链接数据库

default.db=0

#客户端超时时间单位是毫秒

timeout=60000

#最大连接数

maxActive=300

#最大空闲数

maxIdle=100

#最小空闲数

minIdle=1

#最大建立连接等待时间

maxWait=1000

#指明是否在从池中取出连接前进行检验,如果检验失败,则从池中去除连接并尝试取出另一个

testOnBorrow=true

#当调用return Object方法时，进行有效性检查

testOnReturn=true

其他不变，再次测试

**spring整合redis集群（单机只是配置不同）**

**1，加入jar**

  <!-- redis缓存 -->

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-data-redis</artifactId>

        </dependency>

**2,加入配置类：**

package com.aaa.news.util;

import com.fasterxml.jackson.annotation.JsonAutoDetect;

import com.fasterxml.jackson.annotation.PropertyAccessor;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;

import org.springframework.cache.CacheManager;

import org.springframework.cache.interceptor.KeyGenerator;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.redis.cache.RedisCacheManager;

import org.springframework.data.redis.connection.RedisConnectionFactory;

import org.springframework.data.redis.core.RedisTemplate;

import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;

import org.springframework.data.redis.serializer.StringRedisSerializer;

import redis.clients.jedis.HostAndPort;

import redis.clients.jedis.JedisCluster;

import redis.clients.jedis.JedisPoolConfig;

import java.lang.reflect.Method;

import java.net.UnknownHostException;

import java.util.HashSet;

import java.util.Set;

/\*\*

 \* fileName:RedisConfig

 \* description:

 \* author:zz

 \*/

@Configuration

@ComponentScan(basePackages="com.aaa.news.service")

@ConditionalOnClass({JedisCluster.class})

public class RedisConfig {

    public RedisConfig(){

        System.out.println("集群配置加载了。。。");

    }

    /\*\*

     \* 设置缓存对象的序列化方式,不设置会报错

     \* 另外对于json序列化,对象要提供默认空构造器

     \* @param redisTemplate

     \* @return

     \*/

    @Bean

    public CacheManager cacheManager(RedisTemplate redisTemplate) {

        System.out.println("CacheManager...............");

        Jackson2JsonRedisSerializer jackson2JsonRedisSerializer = new Jackson2JsonRedisSerializer(Object.class);

        ObjectMapper om = new ObjectMapper();

        om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);

        om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON\_FINAL);

        jackson2JsonRedisSerializer.setObjectMapper(om);

        redisTemplate.setValueSerializer(jackson2JsonRedisSerializer);

        RedisCacheManager cacheManager = new RedisCacheManager(redisTemplate);

        cacheManager.setDefaultExpiration(300);

        return cacheManager;

    }

   @Bean

    public KeyGenerator myKeyGenerator(){

        return new KeyGenerator() {

            @Override

            public Object generate(Object target, Method method, Object... params) {

                StringBuilder sb = new StringBuilder();

                sb.append(target.getClass().getName());

                sb.append(method.getName());

                for (Object obj : params) {

                    sb.append(obj.toString());

                }

                System.out.println("myKeyGenerator............");

                return sb.toString();

            }

        };

    }

}

**3，启动类加入@EnableCaching**

在配置类(@Configuration)上使用@EnableCaching注解时，会触发一个post processor，这会扫描每一个spring bean，查看是否已经存在注解对应的缓存。如果找到了，就会自动创建一个代理拦截方法调用，使用缓存的bean执行处理。

**4，业务层实现类加入如下配置：**

@Cacheable的使用

@Cacheable是用来声明方法是可缓存的。将结果存储到缓存中以便后续使用相同参数调用时不需执行实际的方法。直接从缓存中取值。最简单的格式需要制定缓存名称。

参数：

value：缓存位置名称，不能为空，如果使用EHCache，就是ehcache.xml中声明的cache的name

key：缓存的key，默认为空，既表示使用方法的参数类型及参数值作为key，支持SpEL

condition：触发条件，只有满足条件的情况才会加入缓存，默认为空，既表示全部都加入缓存，支持SpEL

实例：

@Cacheable(cacheNames="books", key="#isbn")

public Book findBook(ISBN isbn, boolean checkWarehouse, boolean includeUsed)

@Cacheable(cacheNames="books", key="#isbn.rawNumber")

public Book findBook(ISBN isbn, boolean checkWarehouse, boolean includeUsed)

@Cacheable(cacheNames="books", key="T(someType).hash(#isbn)")

public Book findBook(ISBN isbn, boolean checkWarehouse, boolean includeUsed)

@Cacheable(cacheNames="books", key="#map['bookid'].toString()")

public Book findBook(Map<String, Object> map)

condition：

@Cacheable(cacheNames="book", condition="#name.length < 32")

public Book findBook(String name)

@CachePut

如果缓存需要更新，且不干扰方法的执行,可以使用注解@CachePut。@CachePut标注的方法在执行前不会去检查缓存中是否存在之前执行过的结果，而是每次都会执行该方法，并将执行结果以键值对的形式存入指定的缓存中。

@CachePut(cacheNames="book", key="#isbn")

public Book updateBook(ISBN isbn, BookDescriptor descriptor)

@CacheEvict

spring cache不仅支持将数据缓存，还支持将缓存数据删除。此过程经常用于从缓存中清除过期或未使用的数据。

参数：

value：缓存位置名称，不能为空，同上

key：缓存的key，默认为空，同上

condition：触发条件，只有满足条件的情况才会清除缓存，默认为空，支持SpEL,有时候，一个方法可能不适合一直缓存（例如：可能依赖于给定的参数）。属性condition支持这种功能，通过SpEL 表达式来指定可求值的boolean值，为true才会缓存（在方法执行之前进行评估）。

 allEntries 。表示是否需要清除缓存中的所有元素。默认为false，表示不需要。当指定了allEntries为true时，Spring Cache将忽略指定的key。有的时候我们需要Cache一下清除所有的元素。

 清除操作默认是在对应方法成功执行之后触发的，即方法如果因为抛出异常而未能成功返回时也不会触发清除操作。使用beforeInvocation可以改变触发清除操作的时间，当我们指定该属性值为true时，Spring会在调用该方法之前清除缓存中的指定元素。

实例：

@CacheEvict(cacheNames="books", allEntries=true)

public void loadBooks(InputStream batch)

@CacheEvict(cacheNames="books", beforeInvocation=true)

public void loadBooks(InputStream batch

**具体实现类：**

package com.aaa.news.service;

import com.aaa.news.dao.NewsDao;

import com.aaa.news.entity.Dept;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.cache.annotation.CacheEvict;

import org.springframework.cache.annotation.CachePut;

import org.springframework.cache.annotation.Cacheable;

import org.springframework.stereotype.Service;

import org.springframework.web.client.RestTemplate;

import java.util.List;

import java.util.Map;

/\*\*

 \* fileName:NewsServiceImpl

 \* description:

 \* author:zz

 \*/

@Service

public class NewsServiceImpl implements NewsService {

    @Autowired

    private NewsDao newsDao;

  /\*  @Autowired

    private RestTemplate restTemplate;\*/

    @Cacheable(value = "newsList")

    @Override

    public List<Map> getList() {

        System.out.println("..........................");

        return newsDao.getList();

    }

    //@Cacheable(value = "news",key = "#newsId")

    @Override

    public Map getById(int newsId) {

        System.out.println("getById....");

        return newsDao.getById(newsId);

    }

   /\* @Override

    public List<Map> getUserList() {

        return restTemplate.getForObject("http://server-user/getUserList",List.class);

    }\*/

   @Cacheable(value = "newsEList")

    @Override

    public List<Dept> getEList() {

       System.out.println("...newsE..........");

        return newsDao.getEList();

    }

   @CacheEvict(key="#map",value="newsList")

    @Override

    public int save(Map map) {

        return newsDao.save(map);

    }

    @CacheEvict(key="#dept.deptno",value="newsEList")

    @Override

    public int saveE(Dept dept) {

        return newsDao.saveE(dept);

    }

    @CachePut(key="#map.newsId", value="newsEList")

    @Override

    public int update(Map map) {

        return newsDao.update(map);

    }

    @CachePut(key="#dept.deptno", value="news")

    @Override

    public int updateE(Dept dept) {

        return newsDao.updateE(dept);

    }

    @CacheEvict(key="#deptno", value="newsEList")

    @Override

    public int delete(int deptno) {

        return newsDao.delete(deptno);

    }

}

**5，测试效果**

**4、本章总结**

**4.1 总结本章知识点**

**4.2 面试题答案**

**4.3 预习下一章重点**